28丨案例：带宽消耗以及Swap（下）

2020-02-24 高楼

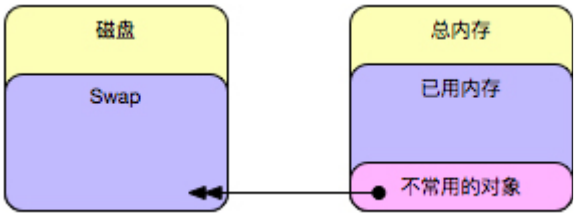
讲述：高楼
时长 28:34 大小 22.90M

上一篇文章我主要分析了带宽消耗，今天，我们来看一下分析的第二和第三阶段，也就是 Swap 分析和数据库分析。

分析的第二阶段

Swap 的原理和对 TPS 的影响

前面有一个扣，是说 swap 多的问题。要理解 swap 为什么是黄的，得先知道什么是 swap。我先画个简易的示意图。



这里先解释一下，对于一个 Linux 系统来说，如果配置并开启了 swap 分区，那么默认的 swappiness 参数是 60。

当上图中已用内存超过 40%（100%-60%）时，系统就会主动切换 swap 和 active 内存中 file 类型的比例。

swappiness 是在内存 reclaim 的时候生效的，而 reclaim 方式同时有两个动作：1. 将 file 相关内存进行回收；2. 将 anon 内存交换到 swap 分区。

所以 swapiness 值越大，swap 分区就用得越多。

对我们现在分析的这个系统来说，来看一下：

```
[root@vhost134 nmon]# sysctl -a|grep swappiness
sysctl: reading key "net.ipv6.conf.all.stable_secret"
sysctl: reading key "net.ipv6.conf.default.stable_secret"
sysctl: reading key "net.ipv6.conf.eth0.stable_secret"
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
vm.swappiness = 30
[root@vhost134 nmon]# sysctl -a|grep vm.min_free_kbytes
sysctl: reading key "net.ipv6.conf.all.stable_secret"
sysctl: reading key "net.ipv6.conf.default.stable_secret"
sysctl: reading key "net.ipv6.conf.eth0.stable_secret"
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
vm.min_free_kbytes = 67584
[root@vhost134 nmon]# swapon -s
文件名                                  类型            大小    已用    权限
/dev/dm-1                               partition       8257532 1048756 -2
[root@vhost134 nmon]# free -m
              total        used        free      shared  buff/cache   available
Mem:           7821        7251         145          27         424         254
Swap:          8063        1040        7023
[root@vhost134 nmon]#
```

我们看到这里配置了一个内存为 8G 左右，已经使用了 7G 多了，swappiness 配置为 30%。也就是说当内存使用超过 8Gx(100%-30%)=5.6G 时，就会使用 swap 分区。

通过 free 看到现在只有 145M 的物理内存剩余，可用内存也只有 254M 了，也就是说现在只有 145（剩余物理内存大小）/7821（总物理内存大小）≈1.9% 的空闲内存，这个比例已经远远小于 swappiness 的 30% 了。

也就是说这系统早就开始用 swap 了。所以上面图中的 swap 飘黄也是很合理的喽！

下面我们就针对应用服务器的 swap 来看是不是可优化。

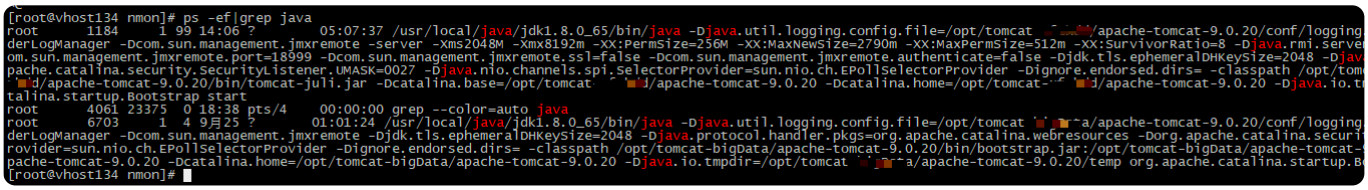
所有人都知道，当 swap 被用的时候，性能肯定会下降，所以在我的测试过程中，一般我都建议把 swap 直接关掉测试性能，有人说这样有什么问题？

那就是没有 swap，让不常用的对象直接占用物理内存，如果物理内存不够用，就把对象删了，后面再创建，这时会增加的是 major fault，那就增加好了，反正是要性能差的。

说得如此硬气，那在生产中怎么办呢？开还是关？有人觉得关了心里有安稳，有人觉得开着心里会安稳。而一个□系统、一个容器、一个节点，如果容量控制的非常好的情况下，我建议关掉。开着它，也只是心里上的安慰，不会有 TPS 处理能力上的提升。

瓶颈分析定位

既然知道了上面的大概原理。对一个运行 Tomcat 应用的服务器来说，那肯定是要先检查一下 JVM 设置为多大。先执行 ps 命令，看一下 Java 进程吧。



关键参数如下：

```
1 JAVA_OPTS="$JAVA_OPTS -server -Xms2048M -Xmx8192m -XX:PermSize=256M -XX:MaxNew
```

JVM 是 1.8.0_65。

这个参数配置有很大的问题。物理内存只有 8 个 G，一个 JVM heap 就配置了 8G，这让其他的东西怎么玩得起来？并且 JDK 是 1.8 了，配置 permsize 是又为啥呢？

虽说有多个地方配置不合理，但是我们也得要知道一下应该配置多少是合理的吧。

看参数的时候，JMX 也配置上了，那就用工具来看吧。

首先来看一下系统资源。先看一下系统资源在压力下的表现：

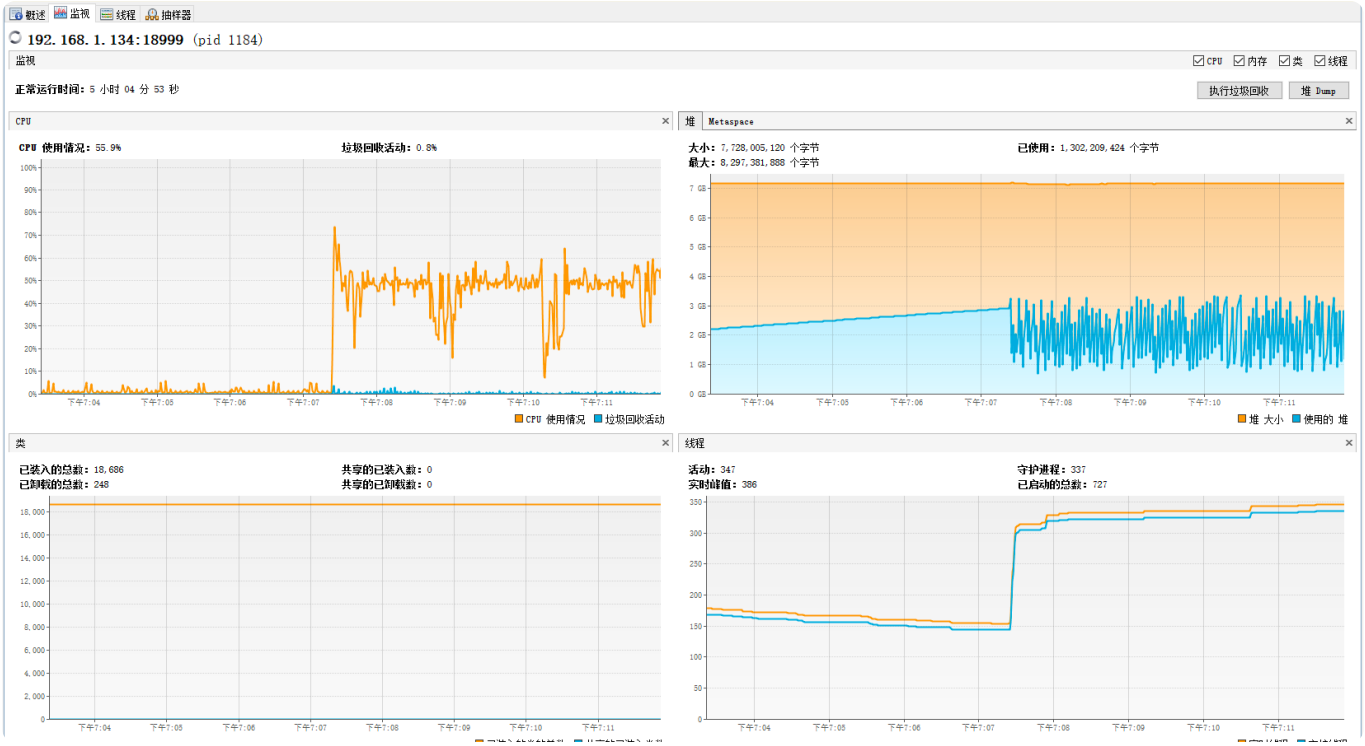
```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
28  0 1067008 144448      0 410976    0    0     1     8    2    2  1  0 99  0  0
29  0 1067008 141328      0 413024    0    0     0     0 22222 25285 54 21 25  0  0
11  0 1067008 139344      0 415360    0    0     0     0 22149 25365 55 21 24  0  0
 7  0 1067008 137724      0 417572    0    0     0     0 22173 25052 54 20 26  0  0
10  0 1067008 134484      0 419632    0    0     0     0 23292 26934 53 21 26  0  0
 8  0 1067008 132792      0 421396    0    0     0 17540 21218 23323 56 22 22  0  0
 5  0 1067008 130488      0 423664   68    0    68     0 20690 23476 55 21 24  0  0
15  0 1067008 129104      0 425400    0    0     0     0 22619 26060 51 22 26  0  0
 9  0 1067008 126036      0 428108    0    0  1028     0 17415 21229 41 22 38  0  0
 7  0 1067008 124324      0 429500    0    0     0     0 15217 19228 31 15 54  0  0
 5  0 1067008 123368      0 430644    0    0    40     0 17994 23110 33 18 48  0  0
 1  0 1067008 122456      0 432120    0    0     0     0 17176 21234 36 17 47  0  0
 1  0 1067008 120812      0 433764   32    0   232    40 15595 19369 35 17 48  0  0
 0  0 1070352 140140      0 417900    0 3416     0  3416 12039 11643 21 13 66  0  0
 3  0 1070352 139148      0 418568    0    0     0     0 14333 18041 28 10 62  0  0
 3  0 1070352 144748      0 419980    0    0    60     0 16132 20688 35 18 47  0  0
 4  0 1068816 135524      0 421360 1188    0  1188     0 12561 15015 28 13 59  0  0
 7  0 1068816 134208      0 423080    0    0     0     0 17640 22815 32 10 59  0  0
 8  0 1068816 133368      0 423852    0    0     0     0 12026 12201 36 16 48  0  0
 0  0 1068816 132612      0 424496    0    0     0     0 11523 14215 21  9 70  0  0
 2  0 1062416 124148      0 425064 6656    0  6656     0 6346 6307 22 12 66  1  0
 2  0 1062416 123104      0 425508    0    0     0     0 15459 19530 27  9 63  0  0
 5  0 1065984 154508      0 410084    0 3616     0  3616 16268 17631 27 16 57  0  0
 1  0 1065984 147308      0 411088    0    0     0     0 13808 16120 34 14 52  0  0
10  0 1065984 139416      0 412784    0    0     0  8468 16214 19656 37 14 49  0  0
 9  0 1065728 137440      0 413856  224    0   336     0 17467 19278 46 17 37  1  0
 4  0 1065728 135924      0 415972    0    0     0     0 19798 21954 55 17 27  0  0
22  0 1065472 133760      0 417420   32    0    32     0 20946 23604 54 20 26  0  0
 9  0 1063680 135832      0 420024 2208    0  2208 12280 21093 22660 58 22 20  0  0
26  1 1063680 128148      0 422284    0    0     0  6964 21623 24668 55 20 26  0  0
 6  0 1063680 125628      0 423588    0    0     0     0 21544 24259 51 24 25  0  0
 9  0 1066752 142912      0 408816    0 2928     0  2928 25051 27590 53 21 26  0  0
 7  0 1066752 140832      0 410812   96    0    96     0 22013 24771 53 24 23  0  0
14  0 1066496 145744      0 413460   32    0    32     0 23377 26667 53 22 25  0  0
 4  0 1066496 136988      0 415512    0    0     0     0 21475 23368 55 23 22  0  0
 4  0 1066496 135708      0 417112   32    0    32     0 21348 23941 56 21 23  0  0
```

1. 队列已经出现，CS 2 万多，in 2 万多，说多不多。我们可以先放着。

2. I/O 没什么□压力，swap 也一直有值，我们要解决的就是它。

3. us：sy 接近 2：1，这个是不良信号，记在心里，后面再说。

其次再看下 JVM 的情况：



CPU 使用在应用上的时间达到 60%，GC 上没耗什么时间，并且从堆的回收□能力上来看，比较正常，只是只用到了 3G 左右，这里有必要给 8G 吗?

线程活动的达到 347，看起来还是在增加的，这里也可能是个问题点，只是现在我们不用关心，它还没跳出来。

从这个 JVM 状态上来看，它完全用不到 8G。在这种状态下，还有另一个 Tomat，并且另一个 Tomcat 中也没有配置 -Xmx -Xms 参数，当没有配置时，默认 -Xmx 是物理内存的 1/4。再加上 thread 用的，物理内存很快就会消耗到 5.6G，所以 swap 飘黄也是吻合的。

优化结果

首先，我们把 JVM 配置成最简，JVM 设置为 4G。

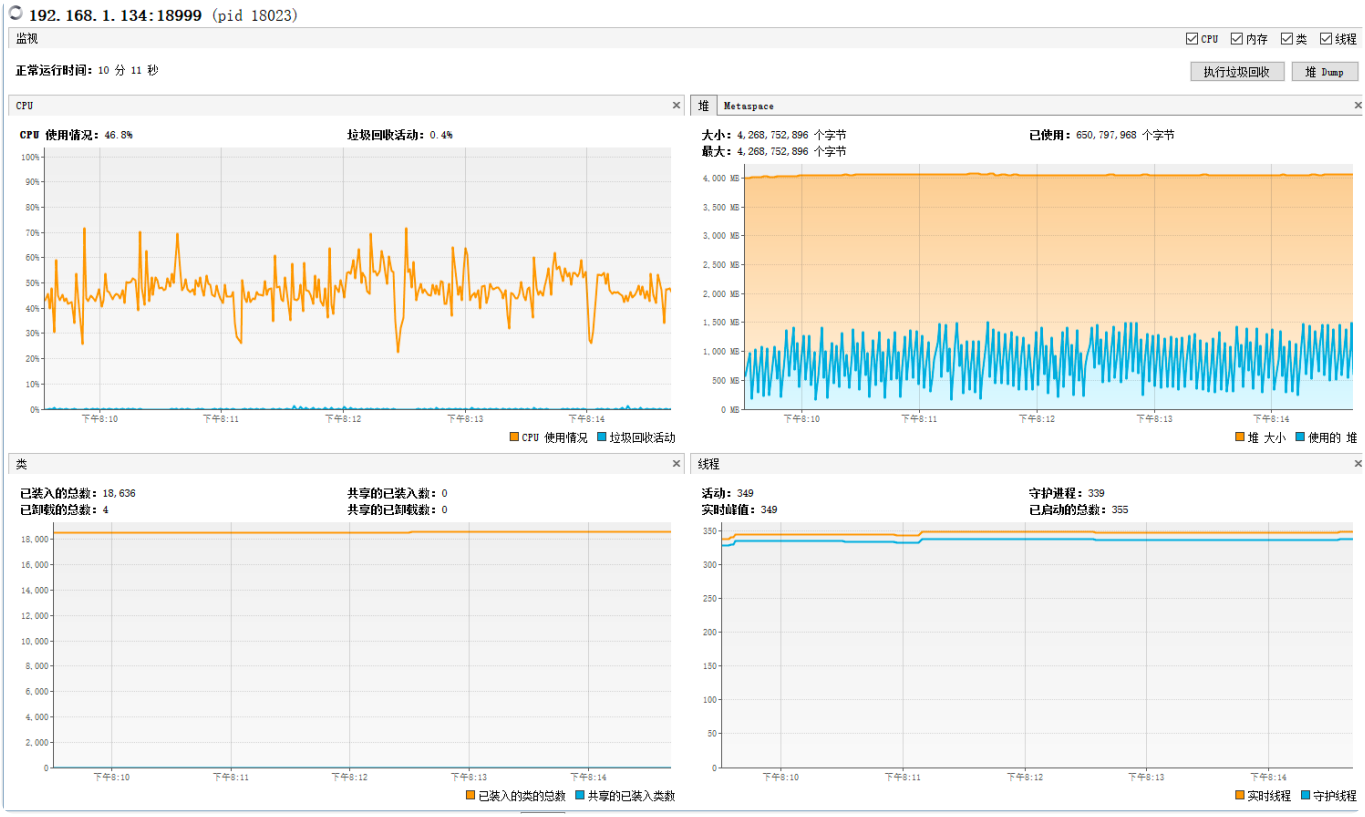
```
1 JAVA_OPTS="$JAVA_OPTS -server -Xms4096M -Xmx4096m"
```

perm 区在 1.8 里都没有了，这几个参数也没啥用。在我的习惯中，MaxNewSize 也是先看要用到多少，再决定配置不配置。有些应用自己不熟悉，也无法直接给出配置，只有测试之后再配置。

各部分配置为多大，都没有定数，要通过测试看需要多少。

而我们现在最重要的是先把性能调整上去，再考虑这些细节内容。这样修改 JVM 就是为了把物理内存使用率低下来，先不修改 swapiness 的比例是为了看下结果，如果用不到 swap 就不再调了，如果还是用了 swap，再来调它。

当我们把 JVM 修改了之后，再执行起来场景。看到内容如下：



CPU 使用率相对前面没有什么变化，但是堆 4G 只用到了 1.5G，可见这个堆连 4G 都用不到。当然我们还是要分析下其他的内容。

还记得我们要解决的是什么问题吧？swap 飘黄了！

从这张图可以看到 Swapping 不□报警了！CPU 占用 70% 左右。说明现在 available 的内存是充足的。

这时我们再看一下系统资源，首先是应用服务器系统资源。

应用服务器系统资源 vmstat 如下：

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
15  0 409600 539944      0 4178704    0    0     1     8    1    3  1  0 99  0  0
18  0 409600 536276      0 4182772    0    0     0     0 22012 20455 58 16 26  0  0
 1  0 409600 531444      0 4187476    0    0     0    28 22573 21583 58 15 27  0  0
13  0 409600 527416      0 4191144    0    0     0     0 23011 24499 56 13 31  0  0
19  0 409600 523652      0 4195424    0    0     0     0 23470 24808 55 16 30  0  0
 7  0 409600 519808      0 4198868    0    0     0     0 22878 22913 62 15 23  0  0
 5  0 409600 513088      0 4202252    0    0     0     0 18644 17825 72 12 17  0  0
 3  0 409600 509556      0 4205460    0    0     0 19888 17641 17965 65 10 24  0  0
 7  0 409600 505704      0 4209396    0    0     0   142 22482 23288 60 13 26  0  0
 9  0 409600 501512      0 4213264    0    0     0     0 24154 24739 59 15 25  0  0
11  0 409600 497876      0 4217332    0    0     0     0 23506 24380 56 14 30  0  0
 8  0 409600 492256      0 4221028    0    0     0     0 22976 24208 56 17 27  0  0
 4  0 409600 489548      0 4225204    0    0     0 89944 22480 22739 54 18 27  0  0
 3  0 409600 485792      0 4229136    0    0     0     0 23764 23280 59 16 25  0  0
 6  0 409600 482296      0 4233108    0    0     0     0 23074 23444 57 15 28  0  0
 9  0 409600 477404      0 4237504    0    0     0     0 23978 23596 60 17 23  0  0
 5  0 409600 473240      0 4240876    0    0     0     0 23695 23902 59 15 26  0  0
15  0 409600 469004      0 4245340    0    0     0     8 23639 24219 59 16 25  0  0
 3  0 409600 465548      0 4248740    0    0     0     0 22918 23793 55 15 30  0  0
11  0 409600 461720      0 4252880    0    0     0     0 22866 22297 59 17 24  0  0
15  0 409600 457132      0 4256984    0    0     0     0 23987 23650 59 17 24  0  0
 9  0 409600 452312      0 4262120    0    0     0     0 22719 21772 62 16 21  0  0
11  0 409600 450052      0 4265420    0    0     0     0 21657 22133 58 14 28  0  0
 9  0 409600 445724      0 4269160    0    0     0     0 24350 24862 57 15 27  0  0
 5  0 409600 441148      0 4273580   64    0    64     0 22912 23653 57 16 27  0  0
 1  0 409600 436800      0 4277312    0    0     0     0 22044 21881 61 14 24  0  0
 1  0 409600 432524      0 4281208    0    0     0     0 22174 21885 58 15 28  0  0
 4  0 409600 429156      0 4284736    0    0     0     0 20157 17712 47 14 39  0  0
11  0 409600 426340      0 4287356    0    0     0     0 18940 16699 48 13 40  0  0
13  0 409600 422616      0 4291456    0    0     0     0 21388 19938 58 15 27  0  0
10  0 409600 429496      0 4295072    0    0     0     0 24267 24777 57 17 27  0  0
12  0 409600 425040      0 4299464    0    0     0     0 23501 24355 60 13 27  0  0
 8  0 409600 417396      0 4303492    0    0     0    20 23808 24591 59 14 27  0  0
 7  0 409600 412296      0 4307688    0    0     0     0 22089 22696 60 14 26  0  0
 4  0 409600 407996      0 4311436    0    0     0     0 22462 23703 52 16 32  0  0
19  0 409600 404180      0 4315120    0    0     0     0 21864 22880 57 15 29  0  0
17  0 409600 399496      0 4318828    0    0     0     0 23926 25294 55 14 31  0  0
 3  0 409600 396656      0 4323036    0    0     0 21636 23287 24358 59 15 27  0  0
13  0 409600 392436      0 4326588    0    0     0    33 23746 24913 58 14 28  0  0
 7  0 409600 388192      0 4331000    0    0     0     0 23561 24543 59 15 26  0  0
29  0 409600 383988      0 4334592    0    0     0     0 23253 24180 58 14 28  0  0
 8  0 409600 381004      0 4338516    0    0     0     0 22050 22698 59 14 27  0  0
```

应用服务器系统资源 top：

```
top - 23:14:52 up 35 days, 13:08,  3 users,  load average: 10.10, 6.55, 3.38
Tasks: 141 total,   3 running, 138 sleeping,   0 stopped,   0 zombie
%Cpu0  : 56.0 us, 12.7 sy,  0.0 ni, 31.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  : 55.3 us, 13.0 sy,  0.0 ni, 31.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 50.9 us, 12.1 sy,  0.0 ni, 23.9 id,  0.0 wa,  0.0 hi, 13.1 si,  0.0 st
%Cpu3  : 56.6 us, 12.2 sy,  0.0 ni, 31.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  8008992 total,   140660 free,  3578388 used,  4289944 buff/cache
KiB Swap:  8257532 total,  7818492 free,   439040 used.  4073076 avail Mem
```

应用服务器系统资源 iftop:

```
TX:             cum:    204MB   peak:   87.2Mb
RX:                     174MB           73.1Mb
TOTAL:                  378MB            159Mb
```

上图中可以看到，对比之前的资源，swap 基本上没有了，CPU 使用率多起来了。但是队列依旧长，sy CPU 消耗还是有点多了。

应用服务器的 si 已经到了 13.1% 了，这个值要关注下，暂时还不能说是问题，但是接着增加下去，肯定会是问题。

网络已经超过 70Mbps 了，峰值上到 87Mbps，这是一个好事，它说明现在处理的业务量确定多了。

接下来是数据库服务器系统资源：

```
 6   0    3336 1164280      20 5315968     0     0     0      0 11456 5837 67 33  0  0  0
88   0    3336 1163860      20 5316004     0     0     0      0 11292 5316 66 28  6  0  0
113   0    3336 1163560      20 5316044     0     0     0      0 11400 5425 68 31  1  0  0
25   0    3336 1164692      20 5315752     0     0     0     18 11370 5286 69 32  0  0  0
32   0    3336 1165324      20 5315788     0     0     0      0 11665 6394 68 33  0  0  0
54   0    3336 1164844      20 5315788     0     0     0      0 11462 6416 71 29  0  0  0
74   0    3336 1164956      20 5315856     0     0     0      0 12171 8159 69 30  1  0  0
57   0    3336 1164868      20 5315912     0     0     0      0 11892 7023 68 32  0  0  0
88   0    3336 1165048      20 5315952     0     0     0     18 12255 8572 67 32  0  0  0
126   0    3336 1164544      20 5315980     0     0     0      0 11511 7640 68 30  1  0  0
89   0    3336 1164568      20 5316020     0     0     0      0 11427 5011 71 29  0  0  0
87   0    3336 1165072      20 5315880     0     0     0      0 11444 5938 71 29  0  0  0
73   0    3336 1164804      20 5315716     0     0     0      0 12151 7968 66 34  0  0  0
32   0    3336 1165204      20 5315804     0     0     0     18 12428 7633 62 38  0  0  0
89   0    3336 1164848      20 5315824     0     0     0      0 11028 7399 72 28  0  0  0
73   0    3336 1164732      20 5315804     0     0     0      0 11995 8208 67 32  1  0  0
11   0    3336 1165056      20 5315856     0     0     0      0 11815 7236 71 29  0  0  0
18   0    3336 1164488      20 5315932     0     0     0    169 11319 6738 68 32  0  0  0
17   0    3336 1163344      20 5316020     0     0     0     18 11932 6959 64 36  0  0  0
23   0    3336 1162032      20 5316200     0     0     0      0 9799 4639 83 17   0  0  0
81   0    3336 1161960      20 5316000     0     0     0      0 9104 5540 83 16   0  0  0
52   0    3336 1162444      20 5315868     0     0     0      0 11204 8595 64 30  6  0  0
20   0    3336 1162716      20 5315876     0     0     0      0 11580 8329 65 29  6  0  0
87   0    3336 1161904      20 5315924     0     0     0     34 11283 4951 70 29  1  0  0
90   0    3336 1161360      20 5315964     0     0     0      0 11307 5464 70 30  1  0  0
45   0    3336 1162236      20 5315988     0     0     0      0 11865 6382 67 33  0  0  0
37   0    3336 1161976      20 5316036     0     0     0      0 11815 8296 68 32  0  0  0
47   0    3336 1161880      20 5316068     0     0     0      0 10791 5319 66 30  4  0  0
108   0    3336 1161580      20 5316108     0     0     0    115 11553 5104 71 29  0  0  0
84   0    3336 1162048      20 5315920     0     0     0      0 10900 5018 72 28  0  0  0
84   0    3336 1162128      20 5315932     0     0     0      0 11690 5407 69 31  0  0  0
56   0    3336 1162100      20 5315976     0     0     0      0 11531 5355 68 32  0  0  0
31   0    3336 1162292      20 5316012     0     0     0      0 10645 4885 70 26  4  0  0
46   0    3336 1161732      20 5316044     0     0     0     46 11290 4916 71 29  0  0  0
22   0    3336 1162884      20 5316076     0     0     0      0 10402 4914 71 25  5  0  0
16   0    3336 1162292      20 5316112     0     0     0      0 11868 5574 70 30  0  0  0
123   0    3336 1161536      20 5316148     0     0     0      0 11251 5279 73 27  0  0  0
```

你可以看到数据库 CPU 都用到这么高了？

# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received...	Sent KB/sec
8390	2859	2706	3824	4387	6023	91	11298	0.00%	13.9/sec	2227.21	6.9
68254	1754	1679	2340	2731	4625	31	10692	0.00%	113.0/sec	934.95	113.8
17122	698	459	1287	1837	3458	21	11608	0.00%	28.5/sec	2296.79	31.7
40	666	536	1574	2683	2965	57	2965	2.50%	10.8/sec	14.03	1.7
62946	378	228	346	538	3256	9	22053	0.00%	104.5/sec	630.98	316.0
156752	1145	933	2293	2845	4513	9	22053	0.00%	259.2/sec	6056.59	466.7

TPS 能到 259.2 了，较之前的 221.5 没有提升多少。但是我们解决了 swap 的问题，还是有了一点点的提升。

那下一个瓶颈在哪里呢？通过上面的数据库资源来看，数据库早就已经被用到了 100% 的 CPU，队列也嗖嗖地涨到了好几十，高的都□超过 100 了。

可见我们在处理□应用服务器的时候，数据库这边□已经早就吃不消了。那下面，我们就先把应用服务器的优化部分放一下，再去分析下一个短板：数据库。

后续性能工作建议

但是这里并不是说应用服务器的优化工作就完成了，还有一些部分需要做的。

1. 优化 JVM 配置参数，至于应该配置成什么值，还需要再测试，可能会有人说，这个测试人员怎么知道呢？请你相信，如果这个值性能□测试人员都测试不出来的话，一般的架构师也不可能知道该设置为多少。
2. 通过监控分析确定 swapiness 的值。
3. 网络带宽又快到占满了，如果 TPS 再提高，网络肯定又支撑不了。

这些扣也都放在这里。因为我们主要是找到系统的短板，并一一解决，才能使整体的 TPS 增加，虽说现在应用服务器上还有优化的空间，但是现在它不是最短的板。

我们在不忘记应用服务器这些问题的同时，再将目光转向数据库。

分析的第三阶段

瓶颈分析定位

先来看看数据库的系统□资源。

```
top - 21:54:53 up 35 days, 11:47,  4 users,  load average: 23.42, 29.04, 22.27
Tasks: 141 total,   3 running, 138 sleeping,   0 stopped,   0 zombie
%Cpu0  : 96.0 us,  3.0 sy,  0.0 ni,  1.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  : 96.0 us,  3.7 sy,  0.0 ni,  0.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 89.7 us,  4.7 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  5.7 si,  0.0 st
%Cpu3  : 97.0 us,  2.7 sy,  0.0 ni,  0.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  8008960 total,  1299620 free,  1391124 used,  5318216 buff/cache
KiB Swap:  8257532 total,  8254196 free,     3336 used.  6093056 avail Mem
 Unknown command - try 'h' for help
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
27448 mysql     20   0 2694540 495032  12908 S 378.4  6.2  41:24.11 mysqld
```

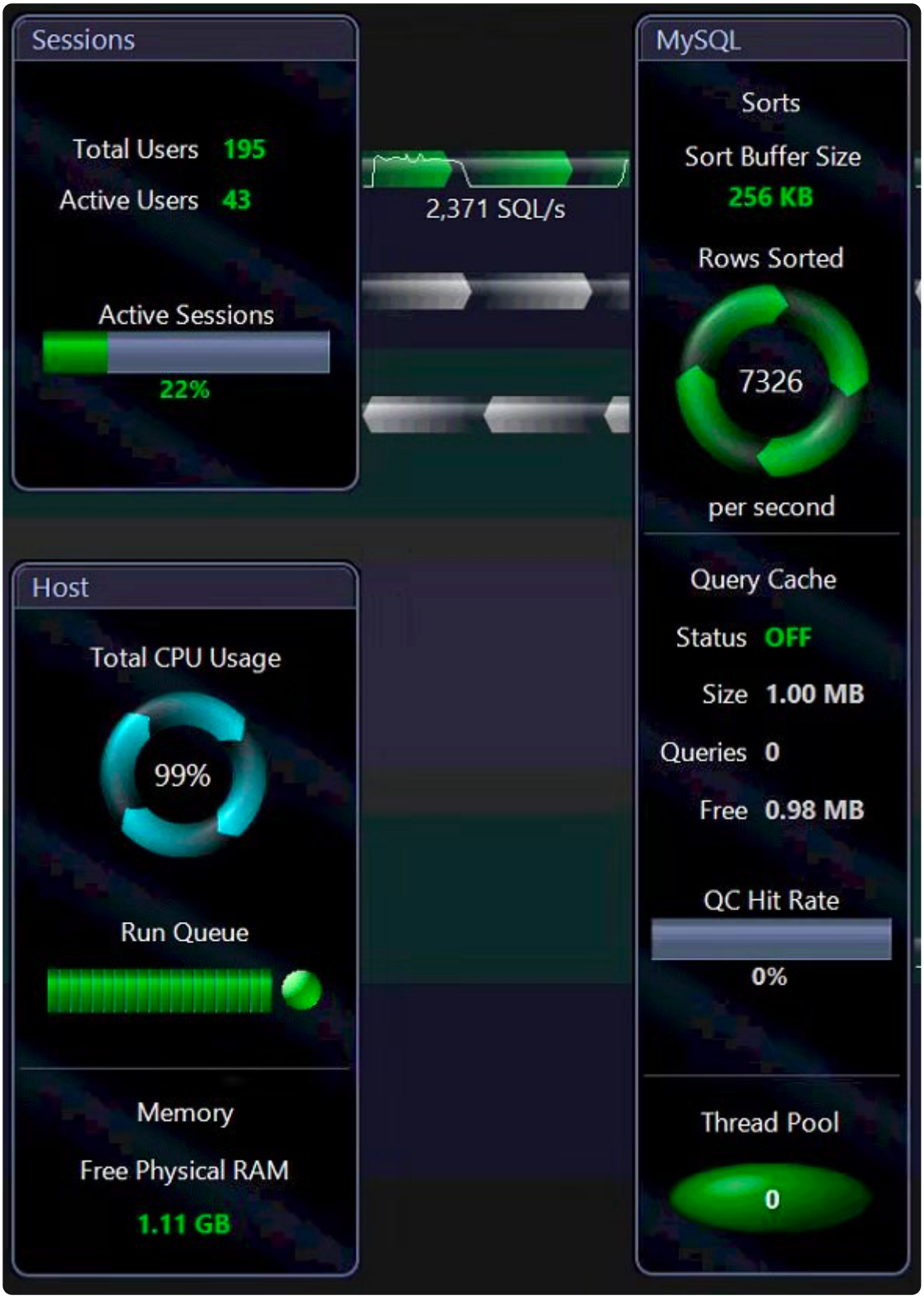
我在很多场合都在强调一个词：证据链。所以基本上分析也会是从 OS 层面开始。

但是证据链这个词说起来容易理解，实际上要想真的有链起来的能力，必须具有基础知识，像分析数据库就更明显。因为当我们不了解系统架构时，想说明一个事情就非常困难。

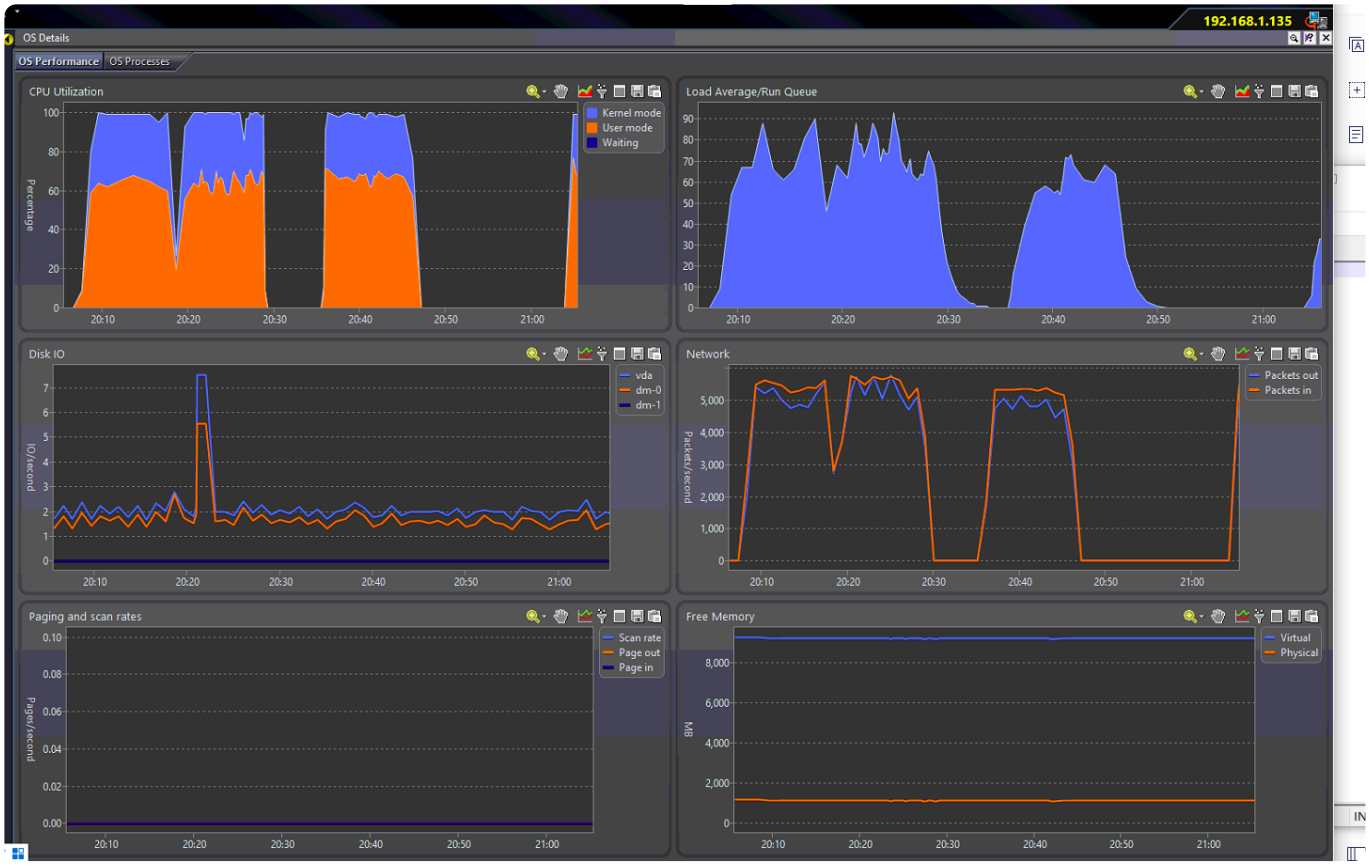
像上面的这个 top，显然 us CPU 使用率非常地高，idle 几乎没有了，只有一个 si 占了 5.7%，这个 si 并不算高，我们在上一阶段看到的应用服务器的 si 都已经达到了 13% 了。

我们说 si 的高或者低，倒不是关键，关键的是它有没有成为我们的瓶颈点。在这个系统中，us cpu 才是我们要关注的重点，因为它实在是太高了。

对于一个数据库来说，要干的事情就是执行 SQL。当分析多了数据库之后，基本上也形成了套路。不管怎么样，还是先看一下基本的监控信息，以下截取一些 Spotlight on MySQL 的有用的图，如果你没有这个工具，用其他的监控工具也是一样的。



从上面的图可以看到，CPU 使用率 99%，Query Cache 是 OFF 的。记下这个位置！



从上图看到，负载队列非常长，但 Disk I/O 没多少，说明队列和□I/O 无关，只是 CPU 的队列，非常好！

Network 也不算大，进出每秒 5000 多个包，我们再来看一下网络用到多少了？

```
TX:              cum:   10.9GB   peak:   58.2Mb
RX:                     2.96GB           12.4Mb
TOTAL:                  13.8GB           69.7Mb
```

峰值也才 70Mbps 左右，即使是 100Mbps 带宽，现在仍然认为有余量（注意！我这里说有余量是因为我同时也检查了网络队列，并没有阻塞，并不是只看了这个值就武断地做了判断）。

通过上面的图可以看到，每秒执行 2500-3000 的 SQL，Sorts per second 达到 800-1000，Sort rows per second 达到 8000-10000。

session 用得倒是也不多，但 Miss Rates 在压力过程中 Query Cache 都是在 100%，并且从最上面的 summary 中可以看到 Query Cache 也是 OFF 的。

为什么没有在看到 Query Cache 是 OFF 的时候就敲黑板呢，这是因为在一些应用中，如果不是查询多的话，这个值 OFF 也不能说有问题，但是在这个应用中几乎所有的语句都是 select，那这个 Query Cache 再不打开就说不过去了呀。这里先记录下这个问题，待会我们的优化动作就是打开 Query Cache。

不管怎么说，对一个数据库来说，主要是执行 SQL 嘛，而对 MySQL 来说，不看 slow log，还能看什么呢。

通过整理 slow log，看到如下内容：

复制代码

```
# Overall: 280 total, 1 unique, 0.59 QPS, 9.53x concurrency _______________
# Time range: 2019-09-26T13:44:08 to 2019-09-26T13:52:06
# Attribute          total     min     max     avg     95%  stddev  median
# ============     ======= ======= ======= ======= ======= ======= =======
```

```
# Exec time          4555s     12s     19s     16s     18s      2s     16s
# Lock time           52ms   130us   662us   185us   273us    53us   167us
# Rows sent             0       0       0       0       0       0       0
# Rows examine      30.81M 112.69k 112.69k 112.69k 112.69k       0 112.69k
# Query size       186.48k     682     682     682     682       0     682


# Profile
# Rank Query ID           Response time    Calls R/Call  V/M   Item
# ==== ================== ================ ===== ======= ===== ===========
#    1 0xBED932B8C940697E 4555.0867 100.0%   280 16.2682  0.16 SELECT test2
```

什么情况？只有 1 unique？0.59TPS？我前面的 TPS 可是有 259.2，□这结果一看就感觉不对。

查看一下 slow_launch_time，配置成了 10s，怪不得看不到慢 SQL。

改 slow_launch_time 为 1s，再跑一遍。看到如下结果：

复制代码

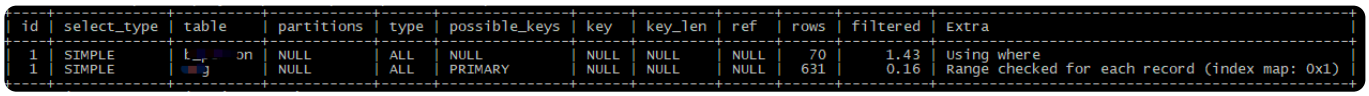
```
# Overall: 620.47k total, 30 unique, 259.39 QPS, 16.76x concurrency ______
# Time range: 2019-09-26T13:44:08 to 2019-09-26T14:24:00
# Attribute          total     min     max     avg     95%  stddev  median
# ============     ======= ======= ======= ======= ======= ======= =======
# Exec time         40082s    88us     23s    65ms   323ms   649ms   332us
# Lock time            61s    30us   363ms    97us   152us     1ms    69us
# Rows sent          1.92M       0     633    3.25   21.45    7.43    0.99
# Rows examine     987.99M       0 112.69k   1.63k  10.29k   5.49k  420.77
# Query size       240.33M      89   1.52k  406.15  833.10  258.19  246.02


# Profile
# Rank Query ID           Response time    Calls  R/Call  V/M   Item
# ==== ================== ================ ====== ======= ===== ==========
#    1 0x160FA75270C56FB8 22892.5442 57.1%  65352  0.3503  0.16 SELECT test1
#    2 0xBED932B8C940697E 15914.4501 39.7%    986 16.1404  0.44 SELECT test2
#    4 0xF0AE7AFA7851C7E8   245.0176  0.6%    175  1.4001  0.11 SELECT test3
#    5 0xFB5A64603A53BFCE    97.2016  0.2%     77  1.2624  0.04 SELECT test4
#   14 0x1E088E88CDC208BE     8.7701  0.0%     14  0.6264  0.23 SELECT test5
```

嗯，这看着顺眼多了。前两个 SQL 占了所有执行时间的 96.8%！第一个 SQL 平均执行时间 350ms，方差 16%。而第二个语句更夸张，平均执行时间 16s，方差 44%。这得收拾！

但是要不要优化这样的 SQL，我们就需要根据 SQL 的分析和业务的分析来判断了。这里我先把执行计划列出来看看。

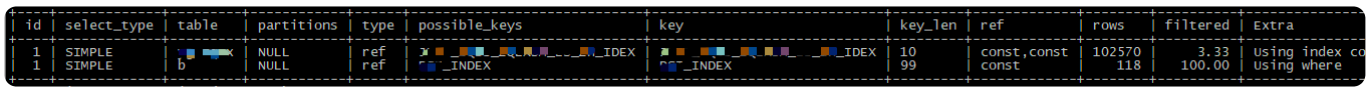
SQL1 的执行计划：



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	[illegible]	NULL	ALL	NULL	NULL	NULL	NULL	70	1.43	Using where
1	SIMPLE	[illegible]	NULL	ALL	PRIMARY	NULL	NULL	NULL	631	0.16	Range checked for each record (index map: 0x1)

1. 没有分区。
2. 不包含子查询或者 union 操作。
3. 全表扫描。
4. 第一个表所查有 70 行，第二个表所查有 631 行，此值仅做为参考，并不精准。
5. 第一个表返回结果只占了读取行数的 1.43%（优化点），第二个表返回结果只占了 0.16%（优化点）。
6. 在第一个表中，Extra 有一个值，using where。
7. 在第二个表中，Extra 有一个值，Range checked for each record (index map: 0x1) 。

SQL2 的执行计划：



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	[illegible]	NULL	ref	[illegible]_IDEX	[illegible]_IDEX	10	const,const	102570	3.33	Using index co
1	SIMPLE	b	NULL	ref	[illegible]_INDEX	[illegible]_INDEX	99	const	118	100.00	Using where

1. 没有分区。
2. 不包含子查询或者 union 操作。
3. 非唯一索引查找，也列出了具体的索引。
4. 第一个表索引列上有 102570 行，第二个表索引列上有 118 行。此值仅做为参考，并不精准。
5. 第一个表返回结果只占了读取行数的 3.33%（优化点），第二个表返回结果占了 100%。
6. 在第一个表中，Extra 有三个值，using index condition; using where; using filesort。

7. 在第二个表中，Extra 有一个值，using where。

这里我要□敲黑板了！！！你是不是不记得 Extra 这些值的含义了？是不是要□祭出你的搜索引擎，要开始查了！

我们这里再来回顾一遍。

using where：对结果用 where 子句中的条件过滤。

Range checked for each record (index map: 0x1)：MySQL 没有找到可以使用的索引，如果前面的表的列值已知，可能会部分使用索引。

using index condition：先条件过滤索引，找到所有符合索引条件的数据行，再用 where 子句中的条件做过滤。

using filesort：Query 中有 Order □By 操作，又无法用索引完成排序，MySQL 不得不选择相应的排序算法来实现。是不是对应上了前面的 sorts per second?

知道了这些基础知识之后，下面再来看一下，两句语句很显然都有优化的空间，尽量使用 filtered 的比例能大一些，至于能不能用到索引，那就看业务的需要了，如果确实是要查很大的索引，表扫还能快点。所以这两个语句，要丢给开发做业务分析了。

优化结果

对数据库，我们有两个优化的方向还记得吧，第一个是 SQL 语句，第二个是 Query Cache。

我们先做第 2 个，将 Query Cache 开启，看一下效果如何。

复制代码

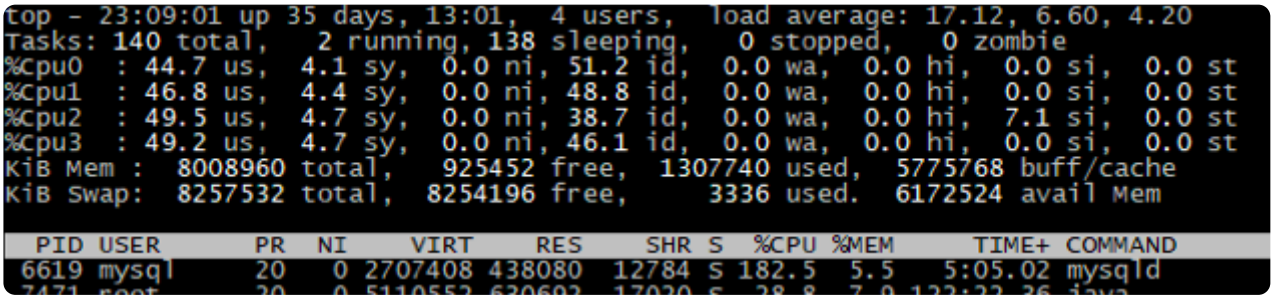
```
1 mysql> show variables like 'query_cache%';
```

查看结果如下：

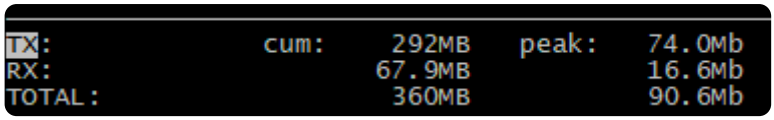
```
query_cache_type  ON
query_cache_size  1048576
```

复制代码

再执行起来场景，看系统资源：

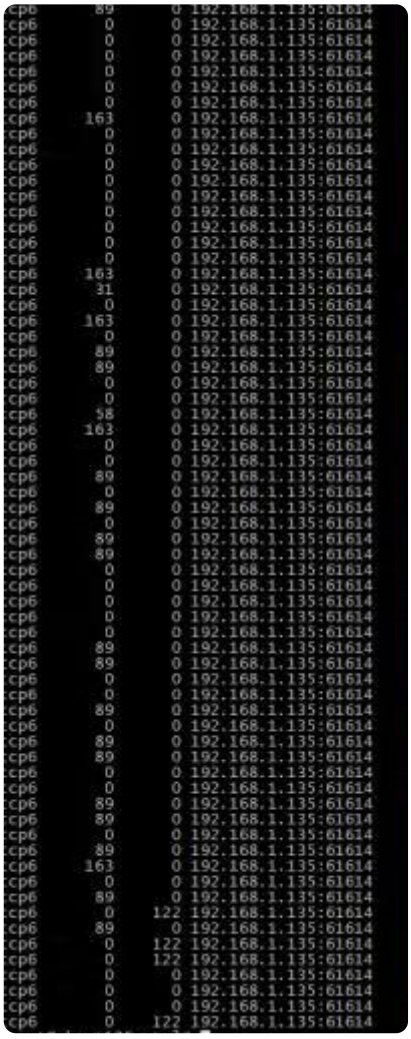


效果还不错哦，us CPU 降到了 50% 以下。

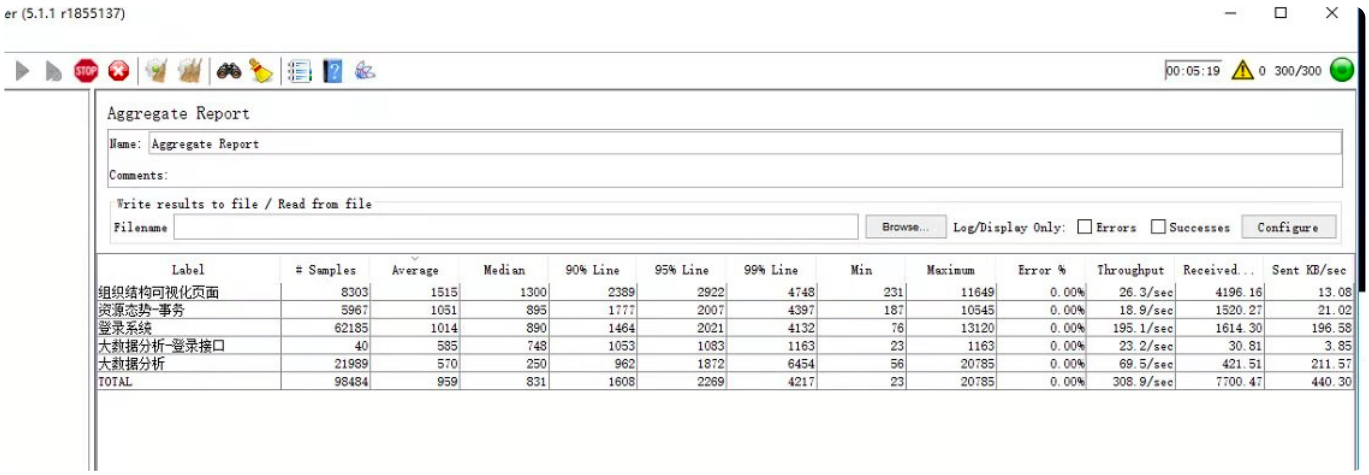


网络峰值时能达到 90Mbps 了，又快把带宽占完了。

查看网络队列：



再检查下队列，这时看到已经有接收队列了。



从 TPS 上来看，现在能到 300 多一点，同时网络接收发送加在一起 8M 左右。

后续性能工作建议

接下来数据库的优化方向就是优化 SQL。

当然还有别的优化建议，我们将在后面再说。

总结

这个案例□从一个概括的描述开始，到各阶段的分析定位，是一个非常完整的过程。从一个项目的角度上来说，现在是不是性能已经达标，要有两方面的判断。

1. 技术方面来说，显然这系统还有很多优化的空间，我们在文中也留了不少的扣。
2. 业务方面来说，系统是否可以上线，就取决于业务指标了。

但是这个性能是不是已经做得完整了呢？显然还没有。现在只是调了一个节点而已。因为这是在测试环境中做的，硬件环境显得非常简单。线上部署结构也会包括□分布式多节点集群等。所以从一个性能项目的角度来说，还远远没有结束。我想如果把这个项目完整地写下来，一本书的容量应该不□为过。

从技术细节上来说，通过几个阶段的具体操作，可以让你有一个性能分析定位的宏观感受，这也是这两篇内容的初衷。性能优化是无止境的，我们要做的是以最少的时间和金钱成本，达到最大的优化效果。

思考题

这一篇文章延续上一篇的分析思路，你能讲一下 Swap 的原理和逻辑，以及分析思路吗？另外，慢 SQL 如何定位出来呢？

欢迎你在评论区写下你的思考，也欢迎把这篇文章分享给你的朋友或者同事，一起交流一下。

课程学习计划

关注极客时间服务号
每日学习签到

月领 25+ 极客币

【点击】保存图片，打开【微信】扫码>>>

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　27 | 案例：带宽消耗以及Swap（上）

精选留言 (2)

写留言

hello
2020-02-24

老师这两篇案例分析特别流畅，一口气看完，思路真的很清晰，感谢老师！

作者回复: 多谢支持，你的收获就是我的动力。

老姜
2020-02-24

mysql用的是什么监控工具?

展开

作者回复: 本篇用的是spotlight。